

## Article

# Rough-Terrain Path Planning Based on Deep Reinforcement Learning

Yufeng Yang <sup>1,2,3</sup>  and Zijie Zhang <sup>1,\*</sup> 

<sup>1</sup> Department of Communication Engineering, School of Automation and Information Engineering, Xi'an University of Technology, No. 5 Jinhua South Road, Xi'an 710048, China; yangyufeng@xaut.edu.cn

<sup>2</sup> Xi'an Key Laboratory of Wireless Optical Communication and Network Research, Xi'an University of Technology, Xi'an 710048, China

<sup>3</sup> Nanjing Institute of Multi-Platform Observation Technology, Nanjing 211500, China

\* Correspondence: 2220320068@stu.xaut.edu.cn

**Abstract:** Road undulations have a significant impact on path lengths and energy consumption, so rough-terrain path planning for unmanned vehicles is of great research importance for performing more tasks with limited energy. This paper proposes a Deep Q-Network (DQN)-based path-planning method, which shapes the reward by introducing a slope penalty function and a terrain penalty function. For the problem of the low exploration efficiency of the  $\epsilon$ -greedy strategy, a hybrid exploration strategy combining stochastic exploration and the A\* algorithm is proposed, after which the agent is trained on rough terrain. The results show that the algorithm can efficiently plan energy-saving paths, converge quickly, and compared with the traditional A\* algorithm and RRT algorithm, performs better under three-dimensional terrain and can choose paths more rationally.

**Keywords:** reinforcement learning; A\* algorithm; path planning; DQN



Academic Editors: Yutaka Ishibashi and Roberto Sacchi

Received: 22 January 2025

Revised: 22 March 2025

Accepted: 28 May 2025

Published: 31 May 2025

**Citation:** Yang, Y.; Zhang, Z. Rough-Terrain Path Planning Based on Deep Reinforcement Learning. *Appl. Sci.* **2025**, *15*, 6226. <https://doi.org/10.3390/app15116226>

**Copyright:** © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

With the continuous development of artificial intelligence [1–6], autonomous driving has become the future trend. Path planning, as the key to autonomous driving navigation, is an extremely important research area in industry and academia. Traditional path-planning algorithms include graph search-based planning and sampling-based planning. Graph search-based algorithms include Dijkstra [7], A\* [8], and their improved algorithms [9,10]. Sampling-based algorithms include Rapidly-Exploring Random Trees (RRTs) [11–13], the Probabilistic Roadmap Method (PRM) [14], etc. In the current era, autonomous driving faces increasingly complex and diverse scenarios. Traditional path-planning algorithms, such as graph-based or sampling-based approaches, are effective in structured environments but often fail to adequately address new challenges arising from high-dimensional, rugged, and dynamic terrains. Issues such as energy consumption optimization, adaptability to complex terrain features, and computational efficiency become critical. The Ant Colony Algorithm (ACO) [15,16] is a typical three-dimensional path-planning method, and its pheromone with certain memory function is a significant advantage, but the three-dimensional path planning of self-driving cars is more likely to encounter uneven, multi-peak and valley, multi-obstruction geographic environment conditions, and it is not able to completely solve the three-dimensional-environment path-planning problem. In addition, intelligent optimization algorithms such as the Genetic Algorithm (GA) [17] and Particle Swarm Algorithm (PSO) [18,19] have been used in rough-terrain path planning. But to adapt to rugged terrain, additional terrain-related costs need to be added to the fitting function,

which makes the search multi-objective and more complex, slowing down convergence. Moreover, the GA and PSO usually incorporate multiple features of the terrain into the cost function, which may lead to a local optimal solution problem due to rough terrain and complex search space.

Regarding rugged-terrain path planning, most studies have focused on obstacle avoidance and safety to ensure that the vehicle does not skid, roll over, etc. For example, considering roll dynamics and actuator dynamics, some scholars have proposed an observer-based vehicle control scheme [20]. Hongqing Tian et al. considered the risk of cross-country terrains and weighed the safety and efficiency of driving to propose a three-dimensional path-planning algorithm based on a kind of potential field-based RRT\* algorithm (PF-RRT\*) [21], which improves the trajectory quality. However, these methods ignore the global characteristics of paths. These studies on rough-path-planning methods for ground vehicles have either focused on accessibility and traveling efficiency or on real-time, localized path planning. Fewer path-planning methods study the tradeoff between the traveling distance and energy consumption under rough road conditions.

The global path planning for rough terrain pays more attention to the energy consumption. For example, Congcai Shen et al. proposed a three-phase global-path-planning algorithm for self-driving vehicles on off-road terrain [22]. Nathan Goulet et al. proposed a global path planner that considers the energy consumption of variable-terrain turns [23], which improves the energy prediction by increasing the turn energy cost with better performance. Haojie Zhang et al. proposed energy-efficient path planning for self-driving ground vehicles based on Ackermann steering based on the A\* search algorithm [24]. Raja and Dutta proposed a path-planning method for wheeled mobile robots in rough-terrain environments, which mainly consists of a combination of the A\* algorithm and the artificial potential field method [25]. A\* can handle different slope levels by adding slope-dependent costs or constraints, but this requires the careful tuning of the heuristic and cost functions. And such traditional algorithms are very time-consuming in large-scale exploration, especially when the map is very large.

Deep reinforcement learning (DRL) is an artificial intelligence method that combines the perceptual capabilities of deep learning and the decision-making capabilities of reinforcement learning. The trained neural network can make decisions based on the system situation and guide the unmanned vehicle in choosing a better path according to the strategy. With the development of deep reinforcement learning algorithms, more and more scholars have begun to use them to solve the problem of path planning. For example, H. Hu et al. proposed a DRL-based navigation method that has been used to navigate robots on flat surfaces with dynamic obstacles for navigation [26]. However, this method only considers flat terrain and does not apply to rugged terrain. For rugged terrain, S. Josef and A. Degani proposed a localized path-planning methodology using a deep reinforcement learning approach for vehicle safety planning [27]. R. O. Chavez-Garcia et al. proposed a learning-based planning method for predicting the passability of mobile ground robot roads on unstructured terrain [28]. Wulfmeier et al. proposed a maximum entropy-based nonlinear inverse reinforcement learning (IRL) framework to more accurately deal with corner situations in the environment, such as staircases, slopes, and underpasses [29]. Zhang et al. used deep reinforcement learning (DRL) to develop and train a DRL network that uses raw perceptual data from the robot's on-board sensors to determine a set of local navigational operations to be performed by the mobile robot, solving the problem of navigating the robot in a cluttered environment with unknown rugged terrain [30]. However, these methods focus on obstacle avoidance and safety, do not consider energy consumption, and are not applicable to global path planning for large off-road terrains.

One of the main challenges in path planning for non-flat terrain lies in the accurate modeling of complex terrain features and the design of effective heuristic functions to guide the search process. Since traditional heuristic algorithms have a limited ability to represent intricate environmental features, it is often difficult to handle non-flat terrain. In this paper, the passability, tradeoff distance, and energy consumption are taken into account. We address this limitation by utilizing the powerful inference capabilities of DRL to perform path planning on rugged terrain. In this paper, a rough-terrain path-planning algorithm called A-DQN is proposed. Compared with traditional search-based global-path-planning algorithms such as A\*, the advantages of A-DQN are that it eliminates the need for heuristic function design on complex terrain, avoids overestimating or underestimating costs, and utilizes the network's reasoning ability, resulting in much shorter decision times than those of traditional search-based global-path-planning algorithms. In addition, during the training period, we used a hybrid training strategy to accelerate the training process. Our main contributions are as follows:

- We consider an unmanned vehicle as an agent and construct a model including states, actions, rewards, and strategies;
- We construct an energy consumption model about the slope and consider the energy consumption and the constraints of different classified terrains through a reward function;
- We used a hybrid exploration strategy combining A\* and stochastic exploration to accelerate the training process. The effectiveness of the algorithm was verified by simulation analysis.

The rest of the sections of our paper are organized as follows. In the second section, the energy consumption model suitable for the agent in this paper is constructed according to the force analysis of the vehicle on the slope. The third section introduces the theory of the DQN algorithm and then introduces our A-DQN algorithm and its applicability on rough terrain for the research problem in this paper. In the fourth section, we describe the simulation and verification of the algorithm. We trained the agent and compared it with a variety of traditional path-planning algorithms on rough terrain to verify the rationality of the proposed algorithm. In the fifth section, we summarize the work of this paper and show the rationality of the exploration strategy proposed by the algorithm in this paper. Compared with the traditional path-planning algorithm, the proposed method has better path quality. The proposed approach is particularly suitable for scenarios involving self-driving cars performing energy-sensitive tasks, such as global-path-planning tasks in mountainous areas, on uneven terrain, or in large, rugged areas.

## 2. Model Construction

Considering that when an unmanned vehicle is traveling in a three-dimensional environment without a road network, its planning path is constrained to go to an uneven, peaky, and obstructed geographic environment, it is especially important to model the slope and energy consumption. In this paper, an unmanned vehicle is considered as an agent. The goal of this paper is to enable the agent to find a reasonable path to reach the target location within a specified step length. By this we mean that it can effectively select the appropriate terrain, including balancing the path length and energy consumption and avoiding difficult terrain. As for the calculation of the energy consumption for rough terrain, the slope is usually considered, and the slope and energy consumption are calculated as follows.

### 2.1. Distance Calculation

In path planning, the Manhattan distance or Euclidean distance is usually used to calculate the distance between two nodes. As this paper is concerned with non-flat terrain,

the Euclidean distance is used to calculate the distance between node  $i$  and node  $j$ . The 3D distance between node  $i$  and node  $j$  is denoted as follows:

$$D_3(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (1)$$

The two-dimensional distance between node  $i$  and node  $j$  is denoted as follows:

$$D_2(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (2)$$

The path length when the whole path has  $n$  sampling points is as follows:

$$D_{total} = \sum_{i=1}^{n-1} D_3(i, i+1) \quad (3)$$

## 2.2. Slope Calculation

In this paper, we plan a path that balances the path length and energy consumption on three-dimensional terrain, so the slope angle is introduced. The slope represents the degree of smoothness of the surface unit and is usually defined as the ratio of the elevation difference to its horizontal distance, and the slope angle is calculated as follows:

$$\theta = \arctan \frac{z_i - z_j}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}} \quad (4)$$

## 2.3. Energy Consumption Model

The main energy consumption of the vehicle during driving comes from the air resistance, tire rolling resistance, gravity, acceleration energy consumption, and braking energy consumption. Assuming that the vehicle travels at a constant speed, the vehicle force analysis is shown in Figure 1; then, we calculate the following:

$$\begin{aligned} F &= F_r + mg \sin \theta + F_a \\ &= f_{rr}(s)mg \cos \theta + mg \sin \theta + \frac{1}{2}\rho A c_D v^2 \end{aligned} \quad (5)$$

where  $F_r$  is the tire rolling resistance;  $F_a$  is the air resistance;  $m$  is the mass of the car;  $f_{rr}(s)$  is the rolling resistance coefficient, and the value in this paper is 0.01;  $g$  is the acceleration of gravity;  $\theta$  is the angle of the gradient;  $\rho$  is the air density;  $A$  is the area of the car's front surface;  $c_D$  is the coefficient of resistance; and  $v$  is the average speed between nodes. Its energy consumption is finally converted into work performed to overcome tire resistance ( $E_r$ ), work performed to overcome gravity ( $E_g$ ), and work performed to overcome air resistance ( $E_a$ ). Then, the energy consumption of a car when traveling a distance of  $d$  uphill and downhill can be expressed as follows:

$$E_{ud} = E_{udr} + E_{uda} + E_g = f_{rr}(s)mg \cos \theta \cdot d + \frac{1}{2}\rho A c_D v^2 \cdot d + mg \sin \theta \cdot d \quad (6)$$

where  $E_{udr}$  and  $E_{uda}$  represent the work performed to overcome tire resistance and air resistance when going uphill and downhill, respectively. The energy consumption of a car when traveling at a constant speed on a flat road can be expressed as follows:

$$E_f = E_{fr} + E_{fa} = f_{rr}(s)mg \cdot d + mg \sin \theta \cdot d + \frac{1}{2}\rho A c_D v^2 \cdot d \quad (7)$$

where  $E_{fr}$  and  $E_{fa}$  represent the work performed to overcome air resistance and tire resistance when driving on a flat road. To eliminate the effect of different vehicles, this paper defines the uphill and downhill unit energy consumption as follows:

$$E_s = \frac{E_{ud}}{E_f} \quad (8)$$

Its unit is u/m, where 'u' is a standardized energy unit defined as multiples of joules, and the unit energy consumption varies for different vehicles going up- and downhill. When traveling at 45 km/h, the work performed to overcome air resistance is about one-quarter of the work performed to overcome tire resistance; i.e.,  $E_{uda} = E_{fa} = \frac{1}{4}E_{fr}$ . Therefore, the unit energy consumption of the uphill and downhill is as follows:

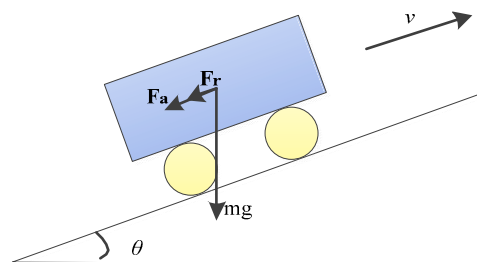
$$E_s = 0.8 \cos \theta + 80 \sin \theta + 0.2 \quad (9)$$

Therefore, the energy consumption between two neighboring nodes  $(i, j)$  can be expressed as follows:

$$E(i, j) = \max(D_3(i, j) \cdot E_s, 0) \quad (10)$$

The total path energy consumption when the whole path has stable  $n$  sampling points is as follows:

$$E_{total} = \sum_{i=1}^{n-1} E(i, i+1) \quad (11)$$



**Figure 1.** Force analysis of unmanned vehicle in uniform motion.

### 3. Path-Planning Algorithm Design

This section introduces the fundamentals of the Deep Q-Network (DQN) algorithm, followed by a detailed presentation of the proposed A-DQN algorithm specifically designed for heuristic exploration in complex terrain environments.

#### 3.1. Description of the Problem

The purpose of path planning is to find a suitable path for the robot to move. The path-planning result path can be expressed as Equation (12):

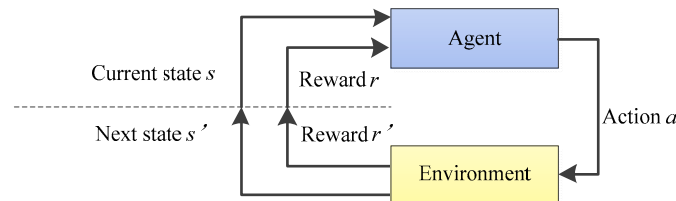
$$path = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_T, y_T, z_T)\} \quad (12)$$

where  $(x, y, z)$  denotes the specific coordinates on the path, and  $T$  denotes the number of search steps. The path-planning task based on reinforcement learning methods requires the process to be modeled as a Markovian decision process (MDP) with Markovian properties, where the MDP consists of tuples  $\langle S, A, P, R, \gamma \rangle$ ,  $S$  is a finite set of states,  $A$  is a finite set of actions,  $P$  is a state transfer probability matrix,  $R$  is a reward function, and  $\gamma \in [0, 1]$  is a decay factor. The basic model of reinforcement learning is shown in Figure 2. The agent interacts with the environment, and at each time step, the agent receives a state  $(s_t)$  from the state space  $(S)$  eventually, the agent chooses to act on the environment by selecting an

action ( $a_t$ ) from the action space ( $A$ ) to enter into a new state ( $s_{t+1}$ ), and the environment feeds back the reward ( $r_t$ ) to the agent. The goal is to enable the agent to learn the optimal strategy ( $\pi$ ) to maximize the future cumulative reward ( $U_t$ ). The policy ( $\pi$ ) is a mapping of states to actions:

$$U_t = r_t + \gamma \cdot \sum_{k=t+1}^n \gamma^{k-t-1} \cdot r_k \quad (13)$$

The path is determined by the search trajectory made by the policy ( $\pi$ ) in  $T$  time steps.



**Figure 2.** The basic model of reinforcement learning.

### 3.2. Principle of the DQN Algorithm

The deep reinforcement learning DQN algorithm combines reinforcement learning and neural networks. The DQN [31] is based on the traditional Q-learning algorithm. The goal of Q-learning is to learn an action-value function that assigns a value to each state–action pair. The goal of the DQN algorithm is to approximate the optimal action-value function ( $Q^*(s, a)$ ) using deep neural networks:

$$Q(s, a, \omega) \approx r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}, \omega) \quad (14)$$

where  $\omega$  represents the neural network weight parameter of the Deep Q-Network. Its loss function is the mean-square deviation of the output value from the target value:

$$L(\omega) = E[(r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}, \omega) - Q(s, a, \omega))^2] \quad (15)$$

The DQN uses Stochastic Gradient Descent (SGD) to update the parameters to achieve the end-to-end optimization objective. The typical DQN uses two networks with the same structure but different parameter update frequencies, an online network for current value estimation and decision making, and a target value network to stabilize the learning process. The parameters of the target network are copied from the current-value network after several rounds of training and are used to compute the target Q-value.

### 3.3. Design of A-DQN Algorithm Based on Heuristic Exploration

#### 3.3.1. State Space

The state space can be divided into four parts, as shown in Equation (20). This paper uses a  $50 \times 50$  map; therefore, each of these four components is represented as a  $50 \times 50$  matrix:

- *Pose* indicates the current position of the agent;
- *Elevation* represents the actual elevation value of each grid point;
- *Goal* indicates the location of the target point;
- *Terrain* is a static representation containing terrain-type labels ( $t_{\text{terrain}}$ ) at each grid point, classified by traversability. In this paper, the grass label is  $t_{\text{terrain}} = 0$  and the mud label is  $t_{\text{terrain}} = 1$ :

$$s_t = \begin{cases} Pose \\ Elevation \\ Goal \\ Terrain \end{cases} \quad (16)$$

### 3.3.2. Action Spaces and Reward Functions

An action space is the set of all possible actions selected by an agent on a map. Action space is divided into discrete action space and continuous action space. This paper pays more attention to the global-path-planning effect of large-scale elevation raster maps, and the DQN works in discrete action space, so we use discrete action space in this paper. The action space of the agent includes eight actions, namely, front, back, left, right, left front, left back, right front, and right back, which are denoted by the numbers 0–7 (i.e.,  $A \in \{0, 1, 2, \dots, 7\}$ ).

In reinforcement learning, a reward function is used to represent how good or bad the selected action is in a given state, helping the agent learn how to optimize its behavior through trial and error to maximize long-term gains. The reward function directly affects the learning process of an agent by guiding it to choose a better behavioral strategy through reward or punishment. A well-designed reward function is the basis for achieving effective learning and optimizing the behavior of an agent.

In this paper, we aim to balance the path length and energy consumption, so we take information including the distance, number of steps, and slope to design the combination reward. In order to ensure that the agent moves closer and closer to the target point, the reward ( $r_1$ ) is set at about the distance from the current point ( $P_c$ ) to the target point ( $P_d$ ), and the larger the distance, the larger the penalty:

$$r_1 = -D_2(P_c, P_d)/50 \quad (17)$$

In order to make the agent travel route smoother and for the tradeoff between the path and energy consumption, a slope penalty is used to evaluate the slope:

$$r_2 = -E(P_c, P_{c+1}) \cdot k_1 \quad (18)$$

The terrain penalty is defined according to the terrain-type label ( $t_{terrain}$ ) as follows:

$$r_3 = -k_2 \cdot t_{terrain} \quad (19)$$

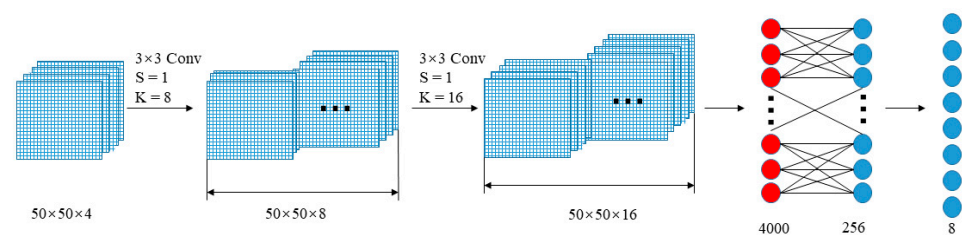
Upon reaching the goal, an additional reward is provided:  $r_4 = 50 \times k_1$ . The total reward *return* is calculated as follows:

$$return = r_1 + r_2 + r_3 + r_4 \quad (20)$$

### 3.3.3. Neural Network Architecture

The neural network used in this paper has a four-layer structure: the first two layers are convolutional layers, and the last two layers are fully connected layers. The structure of the neural network is shown in Figure 3. The convolutional layers are suitable for extracting features from image pixels and matrices, as well as for recognizing certain underlying patterns. The number of neurons must be adjusted with the size of the input matrix, with larger matrices requiring more neurons and smaller matrices requiring fewer neurons. The fully connected layer fits a large amount of input information based on a linear relationship and outputs the final result. The number of neurons in the fully connected layer depends on the number of neurons in the previous layer. The input of the neural network in this

paper is a four-channel graph ( $4 \times 50 \times 50$ ). The four channels represent the elevation map, unmanned vehicle position, end point position, and terrain level. The first convolutional layer contains eight convolution kernels of a  $3 \times 3$  size, which are convolved to obtain eight  $50 \times 50$  feature maps, and the convolution step is 1 and the padding is 1 in the convolutional operation. The second convolution layer contains 16 convolution kernels of a  $3 \times 3$  size, which are convolved to obtain eight  $50 \times 50$  feature maps, and the convolution step is 1 and the padding is 1. The second convolutional layer contains 16 convolutional kernels of a  $3 \times 3$  size, which are convolved to obtain 16  $50 \times 50$  feature maps; the first fully connected layer has 40,000 nodes, and the 16 matrices output from the second convolutional layer are transformed and spliced into a one-dimensional vector of a  $1 \times 40,000$  size; the second fully connected layer contains 256 nodes, and the output layer has 8 nodes representing the Q-values of eight actions.



**Figure 3.** Neural network architecture diagram.

### 3.3.4. Hybrid Exploration Strategy Based on A\*

A\* is the classical heuristic-based path-planning algorithm that determines paths based on heuristic information. The core of the A\* algorithm lies in its evaluation function,  $f(n) = g(n) + h(n)$ , where  $n$  denotes the current node,  $g(n)$  is the actual cost from the starting node to the current node ( $n$ ), and  $h(n)$  is the heuristically estimated cost from the current node ( $n$ ) to the goal node.

The  $\epsilon$ -greedy strategy is often used for random exploration in DQN algorithms. However, for large-scale maps, the training data are huge, and this exploration strategy is randomized, resulting in an increase in the difficulty of the environment. The direct use of traditional DQN algorithms cannot solve the problem of the large training data of neural networks and the difficulty of the environment to be explored, the algorithm convergence speed is slow, and so on. In order to make the exploration both directional and random, this paper adopts the hybrid exploration strategy method (A-DQN) that combines the random strategy and A\*-based exploration strategy, and the specific pseudo-code of the A-DQN algorithm is shown in Algorithm 1. The first 30% of the training rounds take the actions decided by the A\* algorithm with a probability of 0.5, the random decisions are made with a probability of 0.01, and the rest of the actions are decided by the neural network. From 30% to 80% of the training round, the A\* algorithm's decision probability gradually decreases to 0, and in the last 20% of the training round, the A\* algorithm's decision probability always remains 0.

The hybrid exploration strategy integrates heuristic guidance from the A\* algorithm and stochastic exploration from traditional DRL methods. Initially, A\* provides directional guidance, significantly reducing the convergence difficulty. As training progresses, the stochastic exploration component increasingly dominates, allowing for a comprehensive exploration of the state space. This balance effectively addresses the exploration efficiency issues inherent in standard DRL methods, particularly on large-scale terrain.

**Algorithm 1. A-DQN Algorithm pseudo-code**


---

```

1: Initialize environment env and agent
2: for i_episode in range
3:   state, reward, done = env.reset()
4:   while not done and env.steps_counter < max_step
5:     training_strategy ← select_training_strategy(i_episode)
6:     if training_strategy = 'A*' then
7:       action ← select_action_by_A*(state)
8:     else if training_strategy = 'random' then
9:       action ← select_action_random(state)
10:    else
11:      action ← agent.take_action(state)
12:    end if
13:    next_state, reward, done ← env.step(state, action)
14:    replay_buffer.add(state, action, reward, next_state, done)
15:    if replay_buffer.size ≥ minimal_size then
16:      agent.update(replay_buffer.sample(batch_size))
17:    end if
18:  end while
19: end for
20: save agent

```

---

**4. Simulation Experiment**

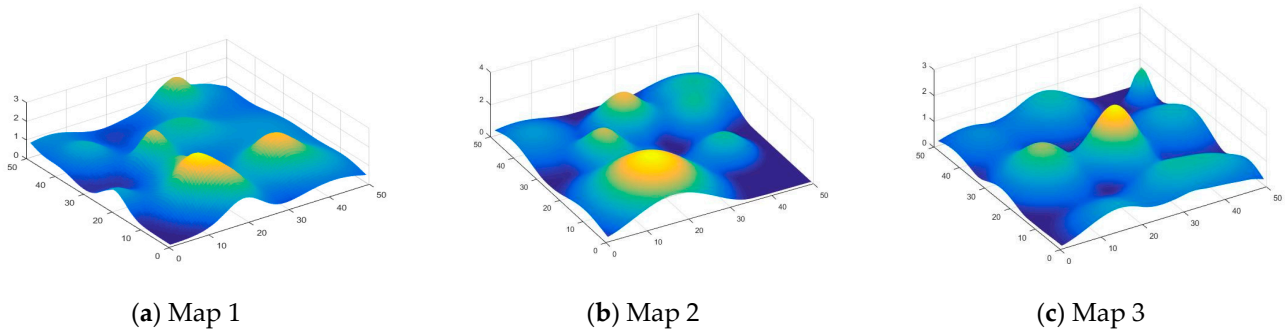
In this section, we describe the simulation experiments conducted to evaluate the performance of the proposed A-DQN method, comparing it with other classical algorithms in terms of computational efficiency and path optimization.

**4.1. Training Setup**

The hardware used in this simulation experiment included Intel® Xeon® CPU, 128 G RAM (Intel Corporation, Santa Clara, CA, USA), and NVIDIA GeForce GTX 1080 Ti (NVIDIA Corporation, Santa Clara, CA, USA). The platform was built based on Gym, and the results are presented in the MATLAB 2016b software environment. In order to verify the performance of the A-DQN algorithm, in this study, the simulation map with a 5000 m × 5000 m area and an elevation value of [0, 300] m was sampled as a 50 × 50 elevation raster map for the experiments; i.e., the x and y axis range was [0, 50], the z-axis range was [0, 3], and the width of the grid point was 100 m. A map of the simulation experiment is shown in Figure 4. After several trials with random starting and ending points in different rounds, the A-DQN algorithm hyperparameters were determined, as shown in Table 1.

**Table 1.** Hyperparameters of the A-DQN algorithm.

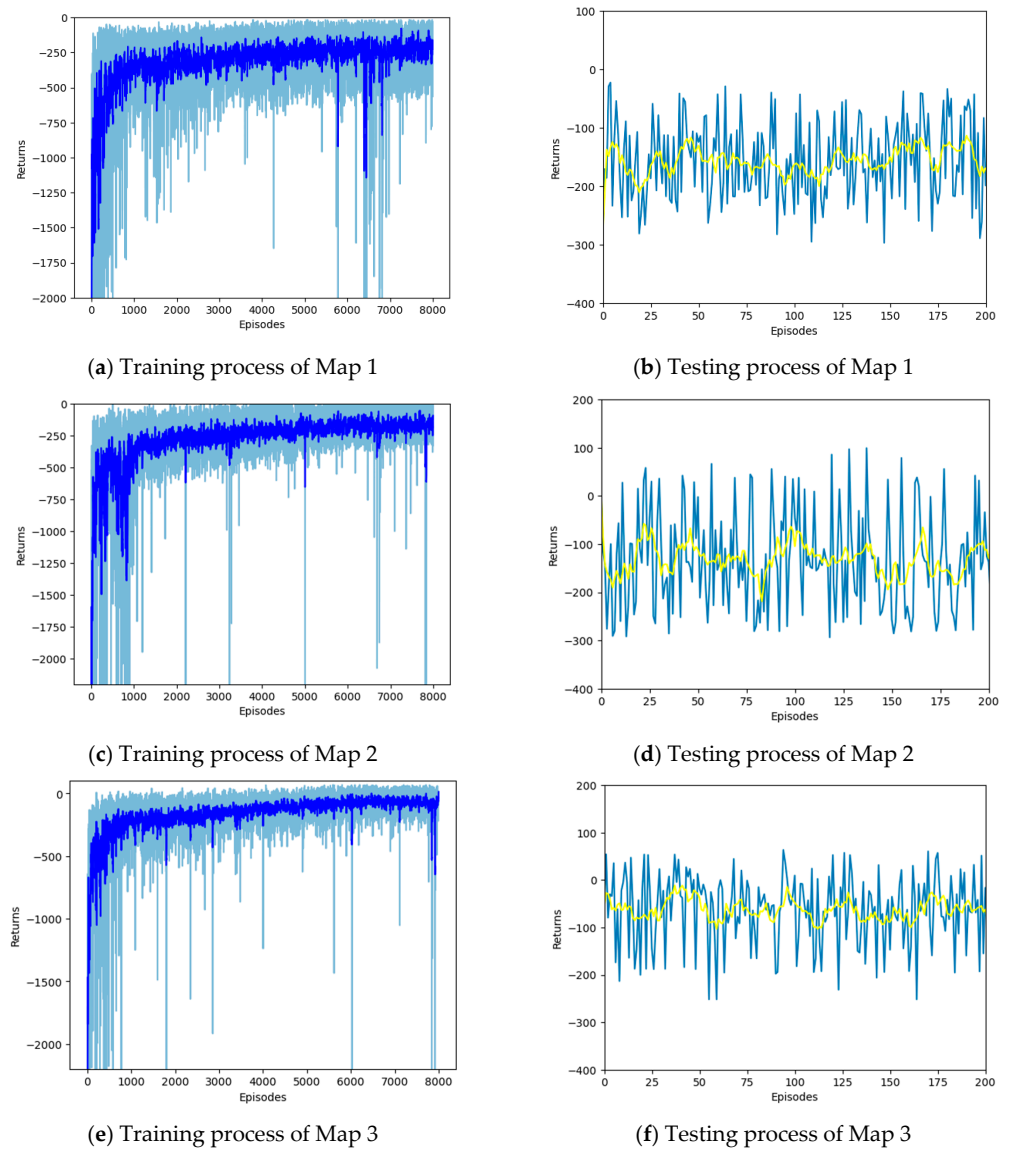
| Parameter     | Meaning                              | Value  |
|---------------|--------------------------------------|--------|
| $\gamma$      | Attenuation factor                   | 0.99   |
| Learning_rate | Learning rate                        | 0.0001 |
| Episodes      | Number of training rounds            | 8000   |
| Replay_memory | Memory pool size                     | 20,000 |
| Max_step      | Maximum number of steps per episodes | 150    |
| Batch_size    | Batch size of memory replay          | 64     |
| $k_1$         | Slope penalty intensity factor       | 4      |
| $k_2$         | Terrain penalty intensity factor     | 10     |



**Figure 4.** Map of simulation experiment.

#### 4.2. Simulation Results and Analysis

In order to verify the path and energy consumption balance effect, the terrain parameters were all set to 0 and the speed was set to 45 km/h at a constant speed for training at first. The variations in the reward with the training rounds of Map 1, Map 2, and Map 3 are shown in Figure 5a,c,e. A total of 200 sets of starting and ending points were randomly selected for testing, and the reward curves of the test rounds are shown in Figure 5b,d,f.



**Figure 5.** A-DQN algorithm reward change curve (light color: original rewards; dark color: rewards smoothed by moving average).

The simulation results of the A-DQN were compared with the RRT, A\* algorithm, and Dijkstra algorithm. The numerical statistics of the results of 200 tests under different maps and different algorithms are shown in Table 2. The cost in the Dijkstra algorithm is computed as the sum of the three-dimensional distance and the energy consumption from the starting point to the current point, with the energy consumption calculated according to Equation (11). In the A\* algorithm, the heuristic function calculates distance based on the intersection line between the vertical plane formed by the current point and the goal point and the terrain surface. The energy consumption along this path is also determined using Equation (11). Since the energy consumption of a path is as important as the distance traveled, a direct comparison is made by summing the two, with paths with smaller sums indicating better global properties.

**Table 2.** Statistics of 200 test results.

| Map   | Algorithm | Mean Distance (m)  | Mean Energy (u)       | Sum          | Mean Time (s)   |
|-------|-----------|--------------------|-----------------------|--------------|-----------------|
| Map 1 | A-DQN     | 3717.2798 ± 282.39 | 11,223.2752 ± 3392.35 | 14,940.5550  | 0.23 ± 0.04     |
|       | Dijkstra  | 4113.0482 ± 257.26 | 11,093.0519 ± 3392.44 | 15,206.1001  | 87.79 ± 0.07    |
|       | A*        | 4109.1041 ± 223.35 | 11,099.4611 ± 3376.78 | 15,208.5652  | 56.08 ± 1.64    |
|       | RRT       | 5255.3713 ± 427.61 | 23,606.0725 ± 3528.53 | 28,861.4438  | 201.41 ± 258.18 |
| Map 2 | A-DQN     | 4139.8562 ± 331.12 | 11,934.2752 ± 3476.42 | 16,074.1314  | 0.23 ± 0.07     |
|       | Dijkstra  | 4451.9524 ± 278.43 | 11,785.6741 ± 3295.34 | 16,237.6265  | 102.12 ± 0.24   |
|       | A*        | 4452.0589 ± 245.31 | 11,788.0265 ± 3338.57 | 11,230.0854  | 65.66 ± 3.06    |
|       | RRT       | 4923.7144 ± 400.08 | 22,933.5654 ± 3427.93 | 27,857.2798  | 242.03 ± 307.56 |
| Map 3 | A-DQN     | 3956.2568 ± 301.13 | 11,697.7548 ± 3428.97 | 15,654.0116  | 0.22 ± 0.03     |
|       | Dijkstra  | 4248.2987 ± 265.70 | 11,516.4835 ± 3219.78 | 15,764.7822  | 97.12 ± 0.03    |
|       | A*        | 4249.9691 ± 230.60 | 11,521.2658 ± 3262.39 | 115,771.2349 | 54.30 ± 1.09    |
|       | RRT       | 4863.5697 ± 395.15 | 20,514.6567 ± 3065.51 | 25,378.2264  | 209.85 ± 248.27 |

From the results, it can be seen that the A-DQN is superfast and the least time-consuming to plan, with an average time of about 0.23 s in the Map 1 scenario. Also, its path quality is comparable to that of the Dijkstra and A\* algorithms. Although the Dijkstra algorithm path quality is high, it takes more time because of the large-scale search. The A\* algorithm takes less time than the Dijkstra algorithm, as it explores through the designed heuristic function. Overall, the A-DQN maintains both the decision time and path quality.

Taking Map 1 as an example, three sets of typical test results were selected to show the comparison of the paths planned using the A\* algorithm, RRT algorithm, and global path planner in the Dijkstra. The paths are shown in Figures 6–8. The quantitative results are shown in Tables 3–5.

**Table 3.** Comparison of test 1 path results.

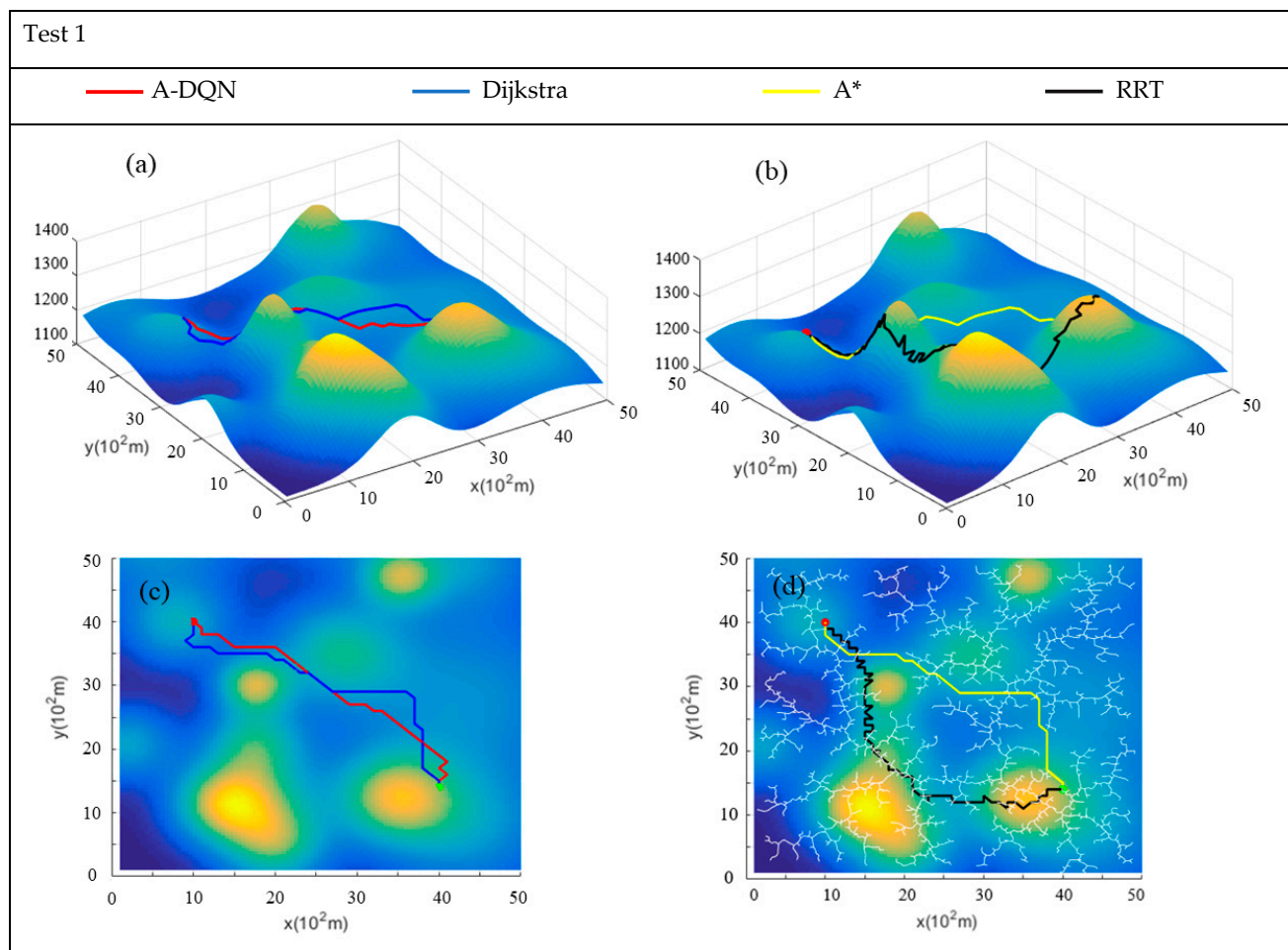
| Test 1     |              |              |                      |          |
|------------|--------------|--------------|----------------------|----------|
| Arithmetic | Distance (m) | Energy (u)   | Sum                  | Time (s) |
| A-DQN      | 4607.9942    | 10,140.0362  | $1.4748 \times 10^4$ | 0.22     |
| Dijkstra   | 5050.3762    | 10,020.6006  | $1.5071 \times 10^4$ | 197.63   |
| A*         | 4850.982     | 10,046.57526 | $1.4898 \times 10^4$ | 167.94   |
| RRT        | 5255.3713    | 25,210.7808  | $3.0466 \times 10^4$ | 213.69   |

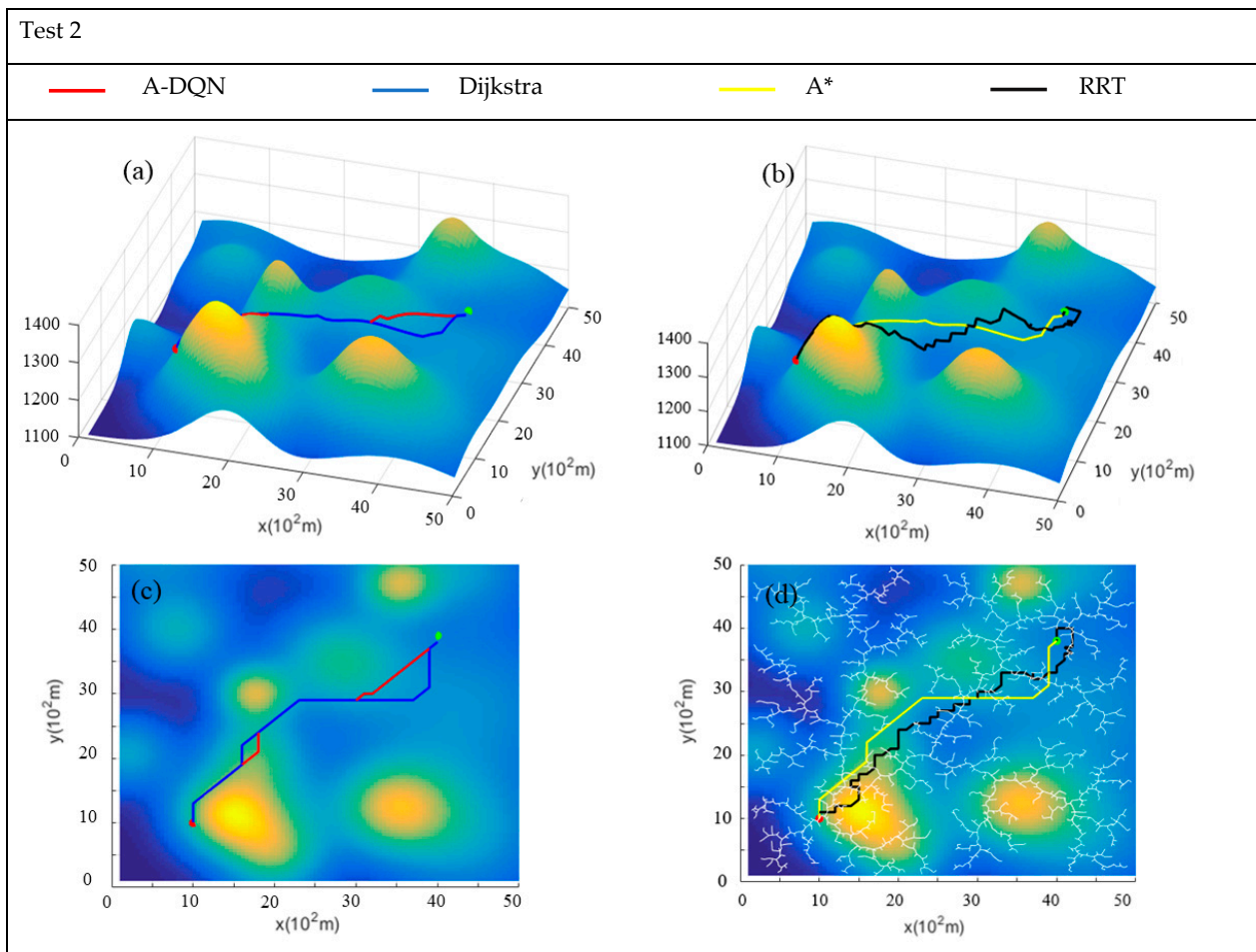
**Table 4.** Comparison of test 2 path results.

| Test 2     |              |             |                      |          |
|------------|--------------|-------------|----------------------|----------|
| Arithmetic | Distance (m) | Energy (u)  | Sum                  | Time (s) |
| A-DQN      | 4518.2799    | 15,580.0503 | $2.0098 \times 10^4$ | 0.23     |
| Dijkstra   | 4868.6638    | 15,319.1458 | $2.0188 \times 10^4$ | 85.72    |
| A*         | 4868.6638    | 15,319.1458 | $2.0188 \times 10^4$ | 42.67    |
| RRT        | 6050.9644    | 30,981.1138 | $3.7032 \times 10^4$ | 158.12   |

**Table 5.** Comparison of test 3 path results.

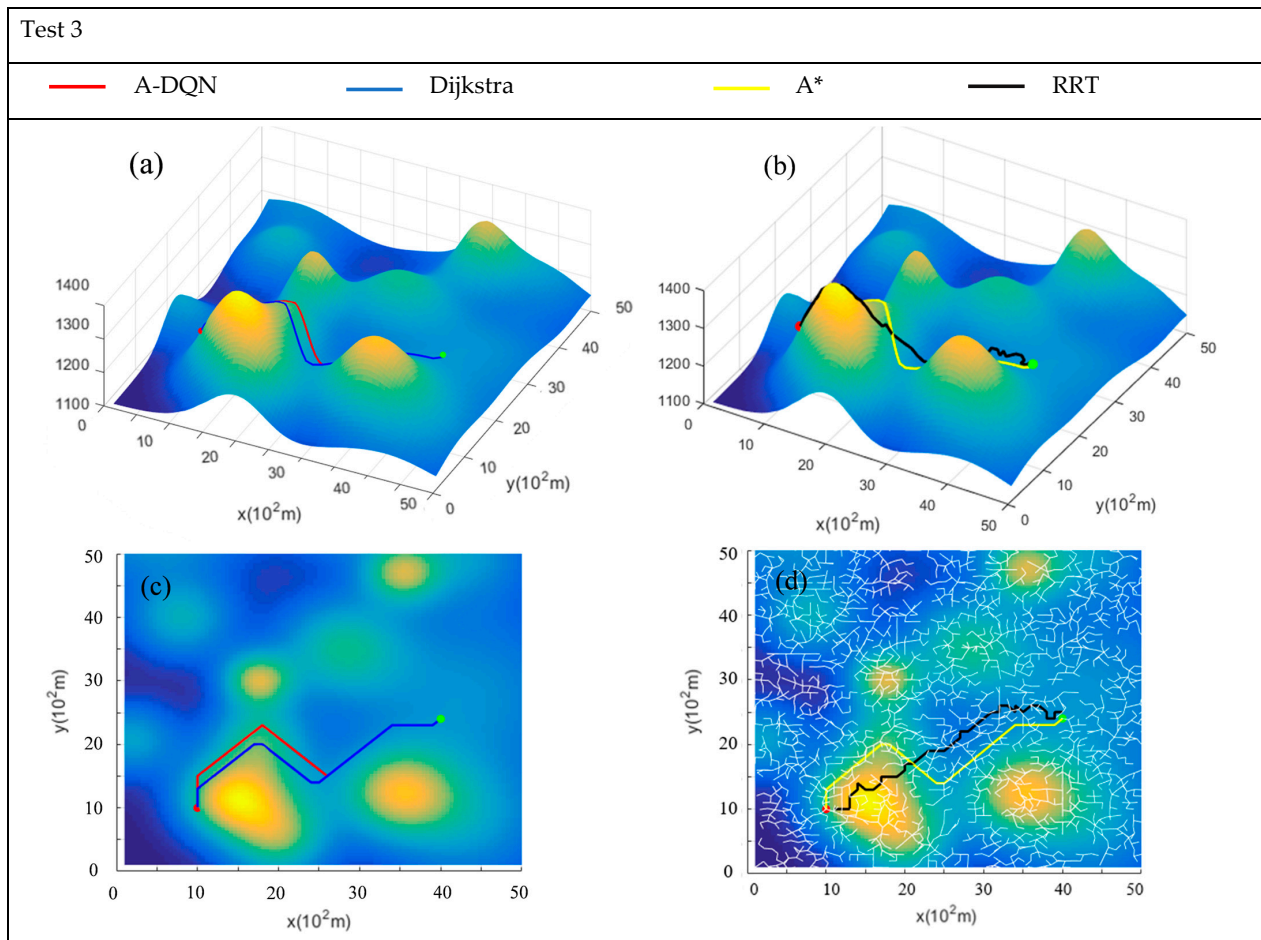
| Test 3     |              |             |                      |          |
|------------|--------------|-------------|----------------------|----------|
| Arithmetic | Distance (m) | Energy (u)  | Sum                  | Time (s) |
| A-DQN      | 4293.9799    | 10,076.9074 | $1.4371 \times 10^4$ | 0.22     |
| Dijkstra   | 4434.5732    | 10,014.551  | $1.4449 \times 10^4$ | 63.60    |
| A*         | 4434.5732    | 10,014.551  | $1.4449 \times 10^4$ | 29.19    |
| RRT        | 4426.41      | 21,454.6406 | $2.5881 \times 10^4$ | 199.36   |

**Figure 6.** Test 1 path diagram: (a) A-DQN (red), Dijkstra (blue) 3D view of the planning path. (b) A\* (yellow), RRT (black) 3D view of the planning path. (c) A-DQN (red), Dijkstra (blue) 2D view of the planning path. (d) A\* (yellow), RRT (black) 2D view of the planning path.



**Figure 7.** Test 2 path diagram: (a) A-DQN (red), Dijkstra (blue) 3D view of the planning path. (b) A\* (yellow), RRT (black) 3D view of the planning path. (c) A-DQN (red), Dijkstra (blue) 2D view of the planning path. (d) A\* (yellow), RRT (black) 2D view of the planning path.

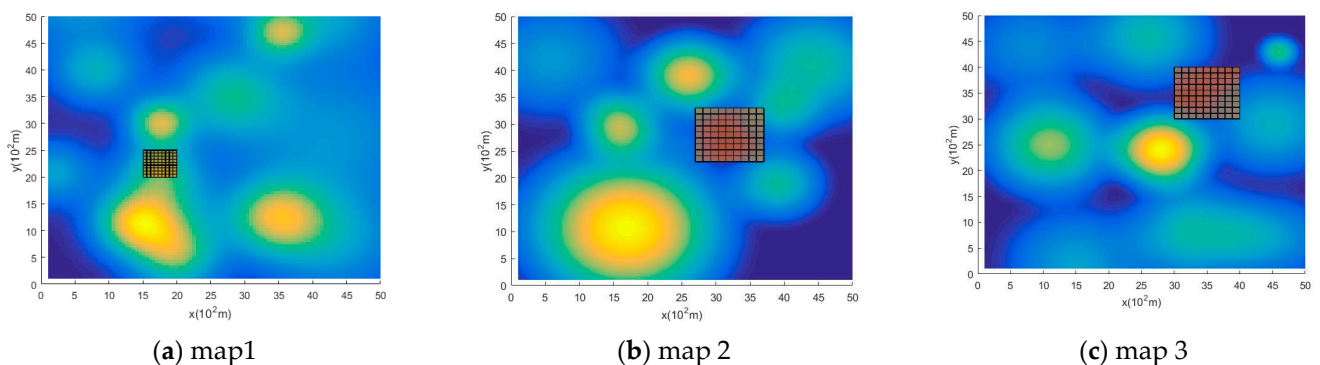
From the quantitative results of the randomized test, as shown in Tables 3–5, it can be seen that the paths planned by the A\* algorithm and Dijkstra algorithm are very close to each other and, in some scenarios, plan the same paths; the RRT algorithm path length and energy consumption are relatively large. The A-DQN algorithm in this paper is very close to the path quality of the Dijkstra and A\* algorithms and, in most cases, prefers to choose the routes with lower gradients and fewer uphill slopes to save energy. As shown in Table 4, the A\* algorithm resulted in a path length of 4868.66 m and energy consumption of 15,319.1458 u. The path planned by A\* is the same as that of Dijkstra. The energy consumption of the paths planned by both is minimal. This is because the global path planner from Dijkstra constructs a cost map, allowing it to plan paths with lower total energy consumption, as shown in Table 5, with the lowest energy consumption at 10,014.551 u. However, the Dijkstra algorithm is very time-consuming due to its large search range. The A\* algorithm has a much shorter planning time compared to the Dijkstra algorithm. As shown in Table 5, A\* takes about half as long as Dijkstra. This is due to the fact that A\* performs a heuristic search. However, the heuristic function in the rugged-terrain design for A\* is complex. The RRT algorithm can cover a large area in high-dimensional space, but its randomness leads to its poor performance in rugged scenes, as seen in Table 3, with a path length of 5255.3713 m and an energy consumption of 25,210.7808 u, both at their largest. It cannot be directly used in the planning of undulating paths and usually consumes more resources.



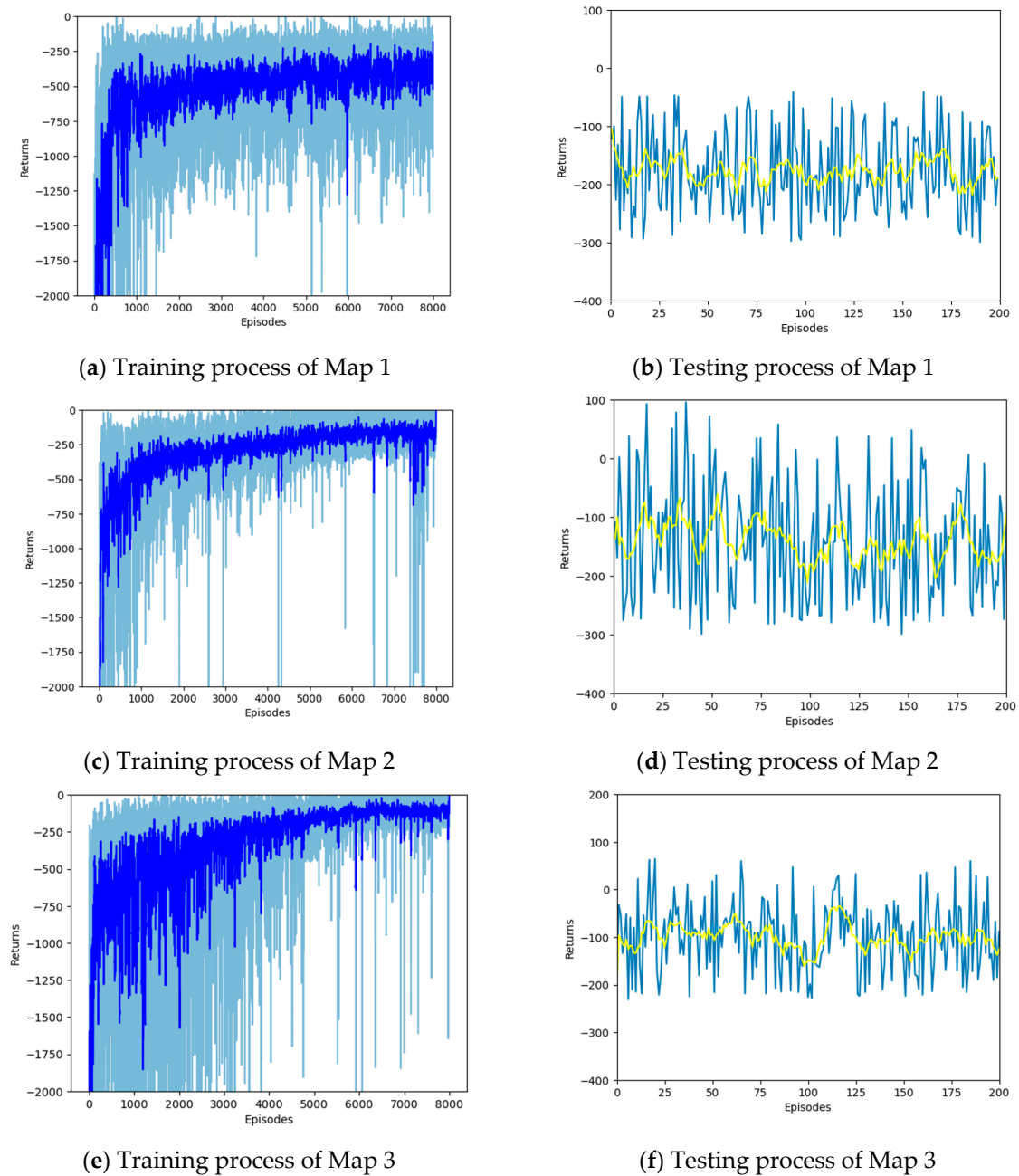
**Figure 8.** Test 3 path diagram: (a) A-DQN (red), Dijkstra (blue) 3D view of the planning path. (b) A\* (yellow), RRT (black) 3D view of the planning path. (c) A-DQN (red), Dijkstra (blue) 2D view of the planning path. (d) A\* (yellow), RRT (black) 2D view of the planning path.

The paths planned by the proposed A-DQN algorithm closely match those of the Dijkstra algorithm, and in some cases, it performs better at balancing the path length and energy consumption. Once trained, the neural network can quickly plan paths through inference, avoiding extensive searching, and improves the efficiency of path planning.

To verify the ability of the terrain recognition, a  $5 \times 5$  range of muddy ground was set in Map 1, and the muddy ground range of  $10 \times 10$  was set in Map 2 and Map 3, as shown in Figure 9. After setting up the muddy-terrain region, the agent underwent 8000 episodes of training. A 200-round test was conducted, and the training convergence curve and test results are shown in Figure 10.

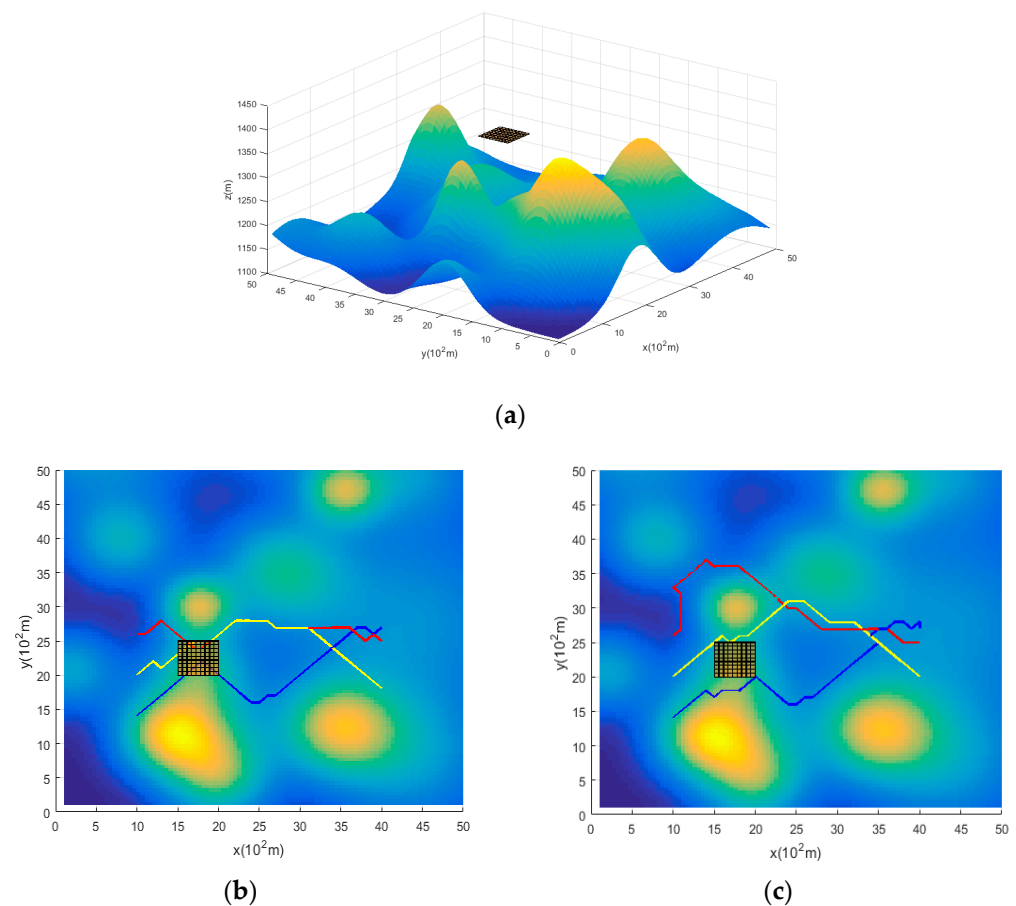


**Figure 9.** Map with  $10 \times 10$  range of muddy ground.



**Figure 10.** A-DQN algorithm reward change curve after muddy-ground setting (light color: original rewards; dark color: rewards smoothed by moving average).

Taking Map 1 as an example, the start point and end point for testing were randomly selected. Several sets of representative starting and finishing points were compared with the training results in the original map, and the results are shown in Figure 11. As can be seen from Figure 11a, the  $5 \times 5$  mud field was set in the valley terrain. The comparison between Figure 11b,c shows that the agent's driving path effectively avoided this  $5 \times 5$  mud range and learned to go around the mud, which indicates that the A-DQN algorithm effectively recognized the terrain. In addition, the agent also chose the route with less uphill in the bypassed driving path. Thus, it is evident that the A-DQN algorithm effectively selects energy-efficient paths and demonstrates reliable terrain recognition, which proves the rationality and effectiveness of the algorithm.



**Figure 11.** Comparison of paths before and after mud land setting of Map 1: (a) three-dimensional view of the muddy area; (b) test path map when mud land parameter is 0; (c) test path map when mud land parameter is 1.

## 5. Conclusions

In this paper, we have studied the global-path-planning problem for unmanned vehicles operating on rough terrains. We propose the A-DQN algorithm to model unmanned vehicles as agents, constructing a model that includes states, actions, rewards, and strategies, and taking into account constraints on energy consumption and different classified terrains through the reward function. The proposed method effectively solves the problem of the difficult convergence of the ground training due to the large size of the search map through the integration of deep reinforcement learning and heuristic exploration. The experimental results demonstrate rapid convergence, effective terrain recognition, and superior path quality compared to the traditional A\*, Dijkstra, and RRT algorithms.

However, the proposed A-DQN method has certain limitations. Firstly, the extensive training process is computationally demanding, particularly for large-scale rough environments. Secondly, the algorithm's performance is sensitive to the parameters chosen in the reward function, which necessitates manual tuning for an optimal performance across different terrains. Our future work will focus on the development of adaptive reward mechanisms that can automatically adjust parameters according to environmental conditions. Future work will include extensive field tests to validate the algorithm on real-world terrains.

**Author Contributions:** Methodology, Y.Y. and Z.Z.; software, Z.Z.; validation, Y.Y. and Z.Z.; writing—original draft preparation, Z.Z.; writing—review and editing, Y.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

**Acknowledgments:** We thank all the participants and all the collaborators who participated in this study.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Ku, D.; Choi, M.; Oh, H.; Shin, S.; Lee, S. Assessment of the resilience of pedestrian roads based on image deep learning models. *Proc. Inst. Civ. Eng. Munic. Eng.* **2022**, *175*, 135–147. [\[CrossRef\]](#)
2. Choi, M.J.; Ku, D.G.; Lee, S.J. Integrated YOLO and CNN algorithms for evaluating degree of walkway breakage. *KSCE J. Civ. Eng.* **2022**, *26*, 3570–3577. [\[CrossRef\]](#)
3. Elasad, Z.E.A.; Mousannif, H. The application of machine learning techniques for driving behavior analysis: A conceptual framework and a systematic literature review. *Eng. Appl. Artif. Intell.* **2020**, *87*, 103312. [\[CrossRef\]](#)
4. Qin, X.; Tang, J.; Liang, J.; Zhu, F. A comparative review on multi-modal sensors fusion based on deep learning. *Signal Process.* **2023**, *213*, 108550.
5. Lee, H.; Kwon, J.; Kwon, C. Learning-based uncertainty-aware navigation in 3D off-road terrains. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 10061–10068.
6. Garcia, F.; Martin, D.; De La Escalera, A.; Armingol, J.M. Sensor fusion methodology for vehicle detection. *IEEE Intell. Transp. Syst. Mag.* **2017**, *9*, 123–133. [\[CrossRef\]](#)
7. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [\[CrossRef\]](#)
8. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1972**, *4*, 100–107. [\[CrossRef\]](#)
9. Bayili, S.; Polat, F. Limited-Damage A\*: A path search algorithm that considers damage as a feasibility criterion. *Knowl.-Based Syst.* **2011**, *24*, 501–512. [\[CrossRef\]](#)
10. Zhang, H.M.; Li, M.L.; Yang, L. Safe path planning of mobile robot based on improved A\* algorithm in complex terrains. *Algorithms* **2018**, *11*, 44. [\[CrossRef\]](#)
11. LaValle, S.M.; Kuffner, J.J. Rapidly-exploring random trees: Progress and prospects. *Algorithmic Comput. Robot.* **2001**, *293*, 303–307.
12. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [\[CrossRef\]](#)
13. Jeong, I.B.; Lee, S.J.; Kim, J.H. Quick-RRT\*: Triangular inequality-based implementation of RRT\* with improved initial solution and convergence rate. *Expert Syst. Appl.* **2019**, *123*, 82–90. [\[CrossRef\]](#)
14. Kavvaki, L.E.; Svestka, P.; Latombe, J.C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580.
15. Vidana, J.I.S.; Motes, J.; Sandstrom, R.; Amato, N. Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robot. Autom. Lett.* **2021**, *6*, 4608–4615. [\[CrossRef\]](#)
16. Reshamwala, A.; Vinchurkar, D.P. Robot path planning using an ant colony optimization approach: A survey. *Int. J. Adv. Res. Artif. Intell.* **2013**, *2*, 65–71.
17. Suresh, K.S.; Venkatesan, R.; Venugopal, S. Mobile robot path planning using multi-objective genetic algorithm in industrial automation. *Soft Comput.* **2022**, *26*, 7387–7400. [\[CrossRef\]](#)
18. Zhang, L.; Zhang, Y.; Li, Y. Mobile robot path planning based on improved localized particle swarm optimization. *IEEE Sens. J.* **2021**, *21*, 6962–6972. [\[CrossRef\]](#)
19. Lv, T.; Feng, M. A smooth local path planning algorithm based on modified visibility graph. *Mod. Phys. Lett. B* **2017**, *31*, 1740091. [\[CrossRef\]](#)
20. Ley-Rosas, J.J.; González-Jiménez, L.E.; Loukianov, A.G.; Ruiz-Duarte, J.E. Observer based sliding mode controller for vehicles with roll dynamics. *J. Frankl. Inst.* **2019**, *356*, 2559–2581. [\[CrossRef\]](#)
21. Tian, H.; Li, B.; Huang, H.; Han, L. Driving risk-averse motion planning in off-road environment. *Expert Syst. Appl.* **2023**, *216*, 119426. [\[CrossRef\]](#)
22. Shen, C.; Yu, S.; Ersal, T. A three-phase framework for global path planning for nonholonomic autonomous vehicles on 3D terrains. *IFAC-Pap.* **2021**, *54*, 160–165. [\[CrossRef\]](#)

23. Goulet, N.; Ayalew, B. Energy-optimal ground vehicle trajectory planning on deformable terrains. *IFAC-Pap.* **2022**, *55*, 196–201. [\[CrossRef\]](#)
24. Zhang, H.; Zhang, Y.; Liu, C.; Zhang, Z. Energy efficient path planning for autonomous ground vehicles with Ackermann steering. *Robot. Auton. Syst.* **2023**, *162*, 106666. [\[CrossRef\]](#)
25. Raja, R.; Dutta, A. Path Planning in Dynamic Environment for a Rover using A\* and Potential Field Method. In Proceedings of the 18th International Conference on Advanced Robotics (ICAR), Hong Kong, China, 10–12 July 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 578–582.
26. Hu, H.; Zhang, K.; Tan, A.H.; Ruan, M.; Nejat, G. Sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain. *IEEE Robot. Autom. Lett.* **2021**, *6*, 6569–6576. [\[CrossRef\]](#)
27. Josef, S.; Degani, A. Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain. *IEEE Robot. Autom. Lett.* **2020**, *5*, 6748–6755. [\[CrossRef\]](#)
28. Chavez-Garcia, R.O.; Guzzi, J.; Gambardella, L.M.; Giusti, A. Learning ground traversability from simulations. *IEEE Robot. Autom. Lett.* **2018**, *3*, 1695–1702. [\[CrossRef\]](#)
29. Wulfmeier, M.; Rao, D.; Wang, D.Z. Large-scale cost function learning for path planning using deep inverse reinforcement learning. *Int. J. Robot. Res.* **2017**, *36*, 1073–1087. [\[CrossRef\]](#)
30. Zhang, K.; Niroui, F.; Ficocelli, M. Robot navigation of environments with unknown rough terrain using deep reinforcement learning. In Proceedings of the 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Philadelphia, PA, USA, 6–8 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–7.
31. Mnih, V. Playing Atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.